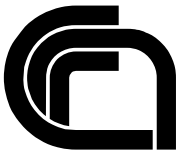

Consiglio Nazionale delle Ricerche

Formal analysis of some secure procedures for certificate delivery

F. Martinelli , M. Petrocchi, A. Vaccarelli

IIT TR-14/2006

Technical report

Dicembre 2006

Istituto di Informatica e Telematica

Formal analysis of some secure procedures for certificate delivery ⋆

Fabio Martinelli, Marinella Petrocchi, and Anna Vaccarelli

Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
Pisa, Italy
{fabio.martinelli, marinella.petrocchi, anna.vaccarelli}@iit.cnr.it

Abstract. The paper describes and formally analyzes two communication protocols to manage the secure emission of digital certificates. The formal analysis is carried out by means of a software tool for the automatic verification of cryptographic protocols with finite behaviour. The tool is able to discover, at a conceptual level, attacks against security procedures.
The methodology is general enough to be applied to several kinds of cryptographic procedures and protocols. It is opinion of the authors that this survey contributes towards a better understanding of the structure and aims of a protocol, both for developers, analyzers and final users.
Key words: Formal analysis, security protocols, digital certificates, computer aided verification.

1 Introduction

In computer science, a protocol is a set of rules, or procedures, for transmitting data between electronic devices, such as computers.

In particular, a security protocol is a specification for transmitting those data in a *safe* way, *e.g.*, avoiding them to be unveiled to unauthorized users or to be modified during their journey from the sender to the intended recipient. These and other goals are generally achieved through the use of cryptography, *i.e.*, the art of writing in secret characters, not comprehensible by anyone but the authorized parties.

The use of cryptographic primitives is by now a standard practice within the area of Internet communication. In the last decades, many researchers have covered various mathematical issues involved in these primitives, leading to a better understanding of the foundations of cryptography. On the other hand, using such well understood cryptographic primitives does not give full guarantees about the fulfillment of the required security properties. Indeed, threats can lie, for example, in the way messages are exchanged over the network.

⋆ Extended and revised version of [1].

A simple example is here informally presented, to show the possible uncertainty about the correctness of a protocol specification. Notation is quite intuitive. However, the reader is invited to see Section 3 for details.

A simple example. Let the reader suppose that A wants to send to the bank $Bank$ an order to move some money to X's account. Thus, A sends the message "*move $1000 to X's account*", signed with its private key, denoted by pk_A^{-1}. (A digital signature is a cryptographic construct aiming at assuring authentication of origin and integrity to a message.)

$$A \mapsto Bank : \{\text{move \$1000 to X's account}\}_{pk_A^{-1}}$$

Since this message is signed by pk_A^{-1}, $Bank$ should be assured that it has been originated by A. Thus, $Bank$ makes the money transfer. Now, let the reader suppose that X eavesdrops on this message. It can pretend to be A and it can send the message again to $Bank$, *i.e.*, :

$$X(A) \mapsto Bank : \{\text{move \$1000 to X's account}\}_{pk_A^{-1}}$$

The signature of this message is still valid. Possibly, X gets $2000. Thus, the protocol has been attacked, even without breaking cryptography.

Starting from these observations, a branch of research in computer security assumes cryptographic primitives to be perfect, and uses a black box view of cryptography. From years, the *formal methods* (and security) community works on these topics.

Formal methods and tools have been successfully applied for the analysis of network security. Exploiting formal methods, the protocol under investigation is described in a given language, then a formal specification of the security property to be analyzed is defined. Whether or not the security property is fulfilled is investigated by formally analyzing the protocol within a hostile environment, *i.e.*, considering the presence of a malicious agent running the protocol together with the honest participants.

In many occasions, formal methods have been proved efficient either to better define the goals of a security protocol (and the mechanisms through which they are achieved) and to offer a rigorous description of the interactions among the participants. Indeed, many examples of correctness of a security protocol – as well as the discovery of attacks on them– may be found in the literature, *e.g.*, [2, 3, 4, 5, 6, 7, 8]. Furthermore, several formal techniques have been developed. Some are based on state exploration, and they can typically ensure error-freeness for bounded systems (*e.g.,* [9, 10, 11]). Other approaches are based on

proof techniques for authentication logic (*e.g.,* [12, 13, 14]). Type systems and other static analyses have also been successfully exploited (*e.g.,* [15, 16]).

Also, automated tools may help in supporting formal theories. They can involve stochastic and timed aspects too. With regard to finite state verification, their development has been so far generally concentrated on two formalisms, process algebras and finite state machine models. Each of the two formalisms has been, and currently is, largely exploited. Here, the main features of two general-purpose verification tools, enriched with support for cryptographic primitives, are briefly discussed. These analysis tools are not the most recent, but they have been extensively used, also to analyze large case studies. They are Murϕ and FDR, respectively [7] and [9].

FDR is a refinement model checker for the process algebra CSP [17]. The basic idea in FDR is to model the correct behavior of the system under investigation as a process in a process algebra, and the protocol specification as another process algebra process. Then, the tool can check if the specification is a refinement of its correct behaviour.

Murϕ is a finite-state machine verification tool. Its verification method is to use state enumeration with state assertion checking. It is an efficient brute-force reachability analyzer.

In both of them (as well as within other frameworks for finite-state verification), a specification is run exhaustively, and in a specific environment. Sometimes, the modeling process helps in finding bugs, even before running the specification by the tool itself. The motivations should be found in the fact that the modeler is forced to think about the behavior of the protocol participants and the intruder in a much more detailed manner than when writing a protocol specification in a non-formal manner. On the other hand, in the past years troubles with lack of crypto-specification/verification education came up. People designing cryptographic protocols may lack expertise, augmenting the risks for a bug-design version of a final product. Automating the verification process has been proved to be really efficient in finding significant attacks, failures as well as weaknesses, to protocols considered correct even for many years. A paradigmatic example is the one of the Needham-Schroeder Public Key protocol, [18], considered correct for a good 17 years after its publication and finally found flawed by Lowe with the support of FDR, [19, 9].

Here, this promising line of research is continued, by presenting a tool for the automated verification of security properties in communication protocols with finite behaviour. In particular, the feasibility of the methodology is shown through two case studies, both inherited from real applications that were born to manage the secure delivery of digital certificates over computer networks.

Digital certificates, [20], are electronic documents linking an identity (*i.e.*, a person or a machine) to a public key. They are issued by a Certification Authority (CA) that can vouch for an individual identity. The way CA vouches for such links is to digitally sign the issued certificate with CA's private key. Typically, a digital certificate contains a public key, information specific to the user (a name, a company, an IP address, etc.), information specific to the issuer, a validity period (starting date - finishing date) and additional management information.

In particular, (part of) the OpenCA software and (part of) the Simple Certificate Enrollment Protocol (SCEP) are considered as two case studies.

OpenCA Labs (http://www.openca.org) is "an open organization aimed to provide a framework for Public Key Infrastructures studying and development of related projects". In particular, the OpenCA developers aim at implementing open source code to easily setup and manage CAs. Consequently, the project provides specification for the whole lifetime of digital certificates, from their emission, through their maintenance, to their expiration. CAs based on OpenCA are really distributed, since there are distinct servers to manage certificates. Furthermore, certificate requests and their validation are performed over the Internet.

SCEP is the evolution of some specification developed by Verisign Inc. and Cisco Systems and it is commercially available in both client and CA implementations. It gives specifications for digital certificate enrollment, access and revocation, for certificates and CRL queries. In particular, SCEP was developed for the distribution of digital certificates to network devices such as routers and gateways (it is indeed its peculiarity, with respect to implementations like OpenCA, to release certificates to machines, rather than to individuals). The following study is based on the SCEP Internet Draft, [21] (that should be considered a work in progress, as mentioned by the same authors of the draft).

CONTRIBUTIONS. The paper offers the following contributions.

– Part of the OpenCA source code and of the SCEP protocol are formally modeled and analyzed. The results of the analysis are here reported.
  With regard to the OpenCA enrollment procedure, it is anticipated that it is correct (at least, at conceptual level). Nevertheless, the attention is focused on the leakage of some sensitive data stored in an on line server. If a malicious adversary discovers these sensitive data, it can force the certification authority to issue erroneous certificates, in which the public key of a honest user is not tied to the identity of the user itself.
  With regard to the security properties listed in the SCEP draft, no attacks are found, at least within an analysis scenario consisting of a finite number of participants. While this does not suffice to ensure the absolute security of the protocol in all circumstances, it does enhance the reliability of SCEP.

However, a vulnerability is noticed concerning the emission of two digital certificates with the same subject name/public key binding.

– The application of automated verification tools is useful to better understand how certain mechanisms and checks ensure certain security features of communication protocols. Generally used with the purpose of verifying a system against a certain (security) property, such automated tools may also offer a valid way to analyse a specification in order to highlight motivation for using a security mechanism rather than another one. Their use might be useful in revealing causes of omitted or erroneously implemented security checks. It might be also useful to render more comprehensive for those less expert some statements written in technical documents, where often it is asserted that a security check is necessary but rarely is the reason given. Thus, in order to understand the reason, some security checks can be omitted in the specification of the protocol. When running the verification tool over the modified specification, an attack can be found, thus revealing the importance of the omitted checks. This methodology extends the use of an automated tool from the common use of verification of a property to a more global aim, *i.e.*, the analysis of various aspects of a system. Furthermore, the same methodology is particularly useful to study security protocols in a formal way by suitably changing the protocol description in order to simulate possible faults and check the relative effect (as is common in so-called methodology "Failure Model and Effect Analysis" adopted in software engineering, see, *e.g.*, http://www.fmeainfocentre.com/ and [22]). In the future, it would be really appealing to systematically create such case studies in the security protocol analysis framework as done in [22] for safety in critical systems.

Finally, to the best of the authors' knowledge, this is the first attempt to give a formal description of some security procedures from OpenCA and SCEP.

The authors do not advocate the cause of their tool more than the one of other existing tools. Being this tool developed within their research team, it is more appealing and practical for them to use it. Nevertheless, it would be interesting to compare the results obtained by using this tool with the ones obtained using other formal instruments.

The paper is organized as follows. In Section 2, a sketch of the adopted analysis approach is presented. Section 3 fixes terminology and notation used throughout the paper. In Section 4, the OpenCA enrollment phase is described and modeled. Section 5 reports the results obtained upon analyzing the OpenCA enrollment phase. In Section 6, the SCEP enrollment phase is described and modeled. Section 7 highlights motivations for the need of some forms of cor-

rectness checks in the SCEP specifications. Section 8 concludes the paper. In the Appendix, Sections A and B give hints to build up a specification file to be analysed.

2 Analysis approach

This paper inherits the analysis approach fully illustrated in [23]. Briefly, it was spurred by the observation that security protocols can be described as *open* systems, *i.e.*, systems with some component following an unspecified behaviour. This may depend on several factors, *e.g.*, one can simply be unable to predict the whole behaviour of a component within a system. Whatever the not specified term is, it is appealing that the resulting system works properly (*e.g.*, satisfies a certain property).

Given the sensitive nature of a cryptographic protocol, one can imagine the presence of a hostile adversary trying to interfere with the normal execution of the protocol in order to achieve some advantage. Due to the unpredictable behaviour of this adversary, this can be seen as an unspecified component of the system under investigation. When considering formal languages for the description of concurrent systems, such as CCS [24] or CSP [17], a concurrent system S with, for example, two specified components A and B and a third unspecified one, can be described as $A \mid B \mid (\_)$, where $\mid$ is the parallel composition operator. The hole takes into account the presence of the adversary, whose behaviour is unpredictable.

Starting from these premises, to formally verify a security property of a specification S implies the verification of the property over S with respect to every possible hole (*i.e.*, every possible adversary behaviour). The problem has been discussed in details in [23], and solved by extending the partial model checking techniques of concurrent systems, [25].

The adversary acts according to a Dolev-Yao model, [26], and it follows a set of message manipulating rules, that are used, *e.g.*, to model cryptographic functions like encryption and decryption. Like the honest participants, it is able to send and receive messages over a set of channels. Also, it can intercept and forge messages. In part, it can derive new messages from the set of messages that it knows at the beginning of the computation. This set is called the intruder's *initial* knowledge. On the other hand, new messages are derived from the intercepted ones, obtained after the beginning of the computation.

Encryption is opaque, *i.e.*, a message encrypted with the public key of *i* cannot be decrypted by anyone but the person who knows the correspondent private key (unless the decryption key is compromised). The adversary can intercept an encrypted message and it can replay the message later, but the structure of the

message is not accessible, *i.e.*, the adversary cannot split the encrypted message unless it knows the decryption key.

The intruder's knowledge grows as the computation evolves. The analysis is over that knowledge, *i.e.*, it is checked if, at a certain point of the computation, that knowledge satisfies a predicate involving a security property. In case of a positive result, the analysis report consists of an attack with respect to that property, *i.e.*, a trace reporting the sequence of actions performed by the adversary and leading to break that property.

The development of the theory has lead to the implementation of a partial model checker, namely the Partial Model Checking Security Analyzer (for short, PAMOCHSA), [27], through which it is possible to analyze distributed systems. As usual, only systems with finite computations will be investigated. This is possible since: 1) the operational language used to specify the protocols does not allow recursion; 2) the messages are of a fixed structure (due to the fact that they are typed, see Appendix A); 3) a finite number of parties and sessions running the protocol are considered; 4) even if the adversary is allowed to generate fresh messages, their structure is subject to the same constraints above mentioned.

It is worth noticing that, though maintaining the analysis over a finite number of parties and sessions, the absence of attacks over a particular system running the protocol does not guarantee that there are no attacks on larger systems running the same protocol. However, [28, 29] show how, under some assumptions, the correctness of a protocol with an unbounded number of parties and sessions can be inferred from the correctness of the same protocol with finite parties and sessions. Hopefully, mixing the approaches may contribute to the development of a fully automated analysis.

The PAMOCHSAtool needs the following set of inputs: the protocol specification; the security property to be checked; the initial knowledge of the intruder. When developing the theory, the operational language Crypto-CCS has been used for specifying the protocols. It is a CCS-based process algebra with extensions for the treatment of cryptographic primitives. Presenting this language is out of the scope of this paper. Full details can be found in [23]. However, the PAMOCHSA input language is not exactly Crypto-CCS. Indeed, it is a variant, a sort of translation of Crypto-CCS into a more readable and intuitive language, that was thought in order to help the common user in preparing a correct input file. An example input file and the PAMOCHSA input language are given in sections A and B of the Appendix.

3 Notation

In this section, terminology and notation used throughout the paper are fixed.

A set of agents able to send and receive messages is considered. The sending and reception of a message is denoted as $i \quad A \mapsto B : msg$, where msg is the exchanged message and i is the i-th communication channel, on which the exchange takes place. A and B are the sender and the receiver of msg, respectively.

The malicious agent is generally denoted as X. X can intercept and also fake messages:

$$\begin{array}{ll} (1) & X(A) \mapsto B : msg \\ (2) & A \mapsto X(B) : msg \end{array}$$

Notation (1) describes X that sends a message msg to B pretending to be A (forgery); (2) denotes: msg, originally intended for B, is actually intercepted by X (interception).

Notation that recurs periodically throughout the paper is:

$$\begin{array}{ll} name_i, pin_i, \ldots := name\, of\, party\, i,\ password\, of\, agent\, i, \ldots \\ pk_i, pk_i^{-1} & := public\, and\, private\, key\, of\, agent\, i \\ \{...\}_{pk_i^{-1}} & := message\, signed\, by\, agent\, i \\ \{...\}_{pk_i} & := message\, encrypted\, by\, public\ key\ of\ agent\, i \\ \{...\}_{KEY} & := message\, encrypted\, by\, symmetric\, key\, KEY \\ h\{m\} & := fingerprint\, of\, message\, m \end{array}$$

The hash functions are a family of functions with the following main properties: i) they take as input a message of arbitrary length and produce an output of a fixed length; ii) they are not-reversible (with high probability); iii) it is computationally infeasible to produce two messages having the same output or to produce any message having a given pre-specified output. Hereafter, the output of a hash function is referred as *fingerprint*.

In the following, it will be often referred to the use of nonces, whose definition is here briefly introduced: a nonce is a parameter that varies with time, *e.g.*, a special marker intended to prevent the unauthorized replay or reproduction of a message. Indeed, a nonce is "generated with the purpose of being used in a single run of the protocol", [19]. Nonces are commonly implemented as pseudo-random strings.

4 The OpenCA Enrollment Phase

OpenCA Labs is an open organization aimed to provide a framework for public key infrastructures studying and development (http://www.openca.org). An

open source code has been developed, and it is being maintained, by the OpenCA developers for the setup and management of Certification Authorities. A formalization of the enrollment procedure of OpenCA will follow.

The following entities are involved in the procedure:

- User (U), requesting a certificate.
- Enrollment Server (ES): a web server used by the users to make certificate requests, import CA certificates, import requested certificates and import other users' certificates. In the investigated implementation, this server is activated on the same machine of the RA Server (they are indeed the same). All the interactions between ES and U are through SSL, [30] (client authentication is not required).
- Local Registration Authority Operator (LRA): a set of trusted operators verifying the correctness of a certificate request. In the investigated implementation, only one operator will be considered.
- Registration Authority Server (RA): the web server to which LRA connects, in order to approve certificate requests. Web connections between LRA and RA are secured through SSL.
- Certification Authority Server (CA): the server where the private key of the Certification Authority is kept. Actually, CA issues certificates. For security needs, this is an off-line server, disconnected by any network. All file transfers (requests/certificates/etc..) with other computers get executed via removable support, *e.g.*, floppies.

Hereafter, a correctly issued certificate (where the name of the user is tied to its public key) is formalized through the name of the user and its public key, both signed by CA's private key, *i.e.*, $\{name_U, pk_U\}_{pk_{ca}^{-1}}$. An abstraction of a digital certificate is here considered, not involving fields like the validity period. Given that the analysis in Section 5 involves the verification of a correct correspondence between an identity and a public key, and it does not involve temporal validity issues, the structure of the certificate has been here simplified for the sake of readability.

Fig. 1 and Fig. 2 give a pictorial representation of the enrollment procedure. Detailed explanation on the use of public keys pk_{c1}, pk_{c2} and pk_{c3} are given in Subsection 4.1. Here, it is anticipated that they serve as a modeling trick to give, at least, confidentiality to the content of the communication.

1. U connects to ES and sends a certificate request consisting of its name $name_U$, a newly created random number pin_U, the public key to be certified pk_U and the so-called Netscape SPKAC, *i.e.*, its public key plus a newly created random number n_U, both signed with U's private key pk_U^{-1}.

$$
\begin{array}{ll}
1 & U \mapsto ES : \{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{c1}} \\
2 & U \mapsto LRA : \{\{name_U\}_{pk_{Gov}^{-1}}, pin_U\}_{pk_{c2}} \\
3.1 & LRA \mapsto RA : \{name_U, pin_U\}_{pk_{c3}} \\
3.2 & RA \mapsto LRA : \{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{c3}} \\
3.3 & LRA \mapsto RA : \{\{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{LRA}^{-1}}\}_{pk_{c3}} \\
4 & RA \mapsto CA : \{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{LRA}^{-1}} \\
5 & CA \mapsto RA : \{name_U, pk_U\}_{pk_{ca}^{-1}} \\
6 & ES \mapsto U : \{name_U, pk_U\}_{pk_{ca}^{-1}}
\end{array}
$$

Fig. 1. The OpenCA enrollment procedure.

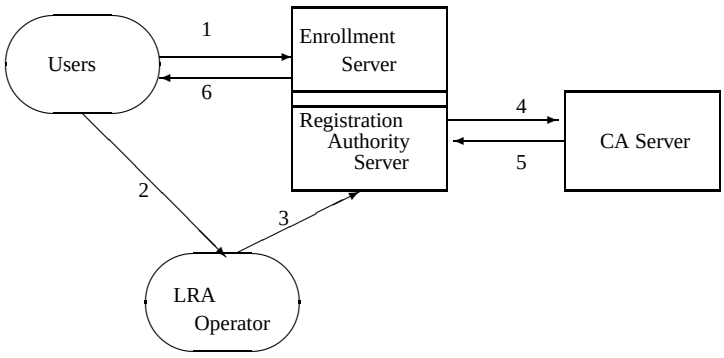

Fig. 2. Graphical representation of the OpenCA infrastructure.

All these data are stored in the RA server (note that in the investigated implementation ES and RA coincide), waiting for being validated. SPKAC acts as a Proof of Possession (POP), attesting that U, that is requesting to link its name to a public key, also owns the corresponding private key. (Only someone who knows the private key can create that SPKAC.) n_U is inserted to prevent replay attacks. It is up to the recipient of the message to store n_U in order to detect later replays. In the OpenCA specification, this message is sent across SSL. For the rendering of the transmission over SSL the reader is invited to read the following Subsection 4.1, *OpenCA model*.

2. U personally reaches the LRA Operator to identify himself by means of a valid paper, *e.g.*, an identity card, and by showing the same pin already present in the formulated request. The identity card is here represented by $name_U$ signed by the private key of the Government, *i.e.*, $\{name_U\}_{pk_{Gov}^{-1}}$. It is assumed that it cannot be forged. Furthermore, even if the structure of the ID card has been simplified in its formal representation, it is assumed that it contains the photo of the owner, as usual.
3. LRA connects to RA and approves the request corresponding to $name_U$ and pin_U. In particular, LRA transmits to RA a query regarding a particular pair identity-pin (step 3.1). Then, RA sends back the request received in step 1 (step 3.2) (corresponding to that pair identity-pin). Finally, LRA sends the request back signed with its private key pk_{LRA}^{-1} (step 3.3). Signing the request means that LRA has personally identified who is requesting the certificate and that there is a correspondence between name and pin received in step 2 and what received in step 3.2. All these three steps are through SSL. Again, the reader is referred to the next subsection for details.
4. The request is exported from RA to CA (through removable media).
5. Before issuing the certificate, CA operator must verify the presence of a correct LRA signature over the request. Then, CA operator checks the correctness of SPKAC (by applying pk_U that is present in message 4). If the checks are satisfied, the certificate is issued and exported into RA (through removable media).
6. Finally, U connects to ES and gets its certificate.

4.1 OpenCA model

All the interactions with the Certification Authority Server (steps 4 and 5) are via removable media through trusted operators. No adversary may intercept or listened to the exchanged messages (at least with high probability).

As far as the other steps are concerned, some modeling assumptions of the OpenCA enrollment are hereafter listed.

– In the OpenCA implementation, steps 1, 3.1, 3.2, 3.3 are performed through SSL connections. That means that data sent over the network are kept confidential (*i.e.*, only the intended recipients can understand the meaning of those messages), and that the server identity is somehow authenticated to prevent server spoofing.
  Step 1 requires no client authentication, *i.e.*, only the user must be assured that the other party is actually the enrollment server to which it is to request a certificate.
  On the contrary, steps 3.1, 3.2 and 3.3 require both client (LRA) and server (RA) authentication, *i.e.*, both RA and LRA must be assured about the identity of the interlocutor.
  To completely model the overall architecture, the SSL protocol should be encoded within the formalization. To avoid the entire encoding, some confidential channels established between the interested parties are modeled. In particular, if one supposes that all possible users (and even the adversary) know a public key pk_{c1} and that only ES knows the corresponding private key pk_{c1}^{-1}, encrypting requests in step 1 with pk_{c1} guarantee their confidentiality. One may observe that the same does not guarantee server authentication. Given that the analysis is here focused on the right correspondence between a public key and a name in a issued certificate, and since it is not concerned with specific SSL properties, the claim is that one can afford to maintain the formalization to a lower level of granularity with respect to the SSL protocol. Moreover, if no attack is discovered with a *weaker* model, one may suppose that no attack would be discovered with a stronger representation (with respect to the same analyzed properties). Thus, it is assumed that everybody knows pk_{c1} and only ES knows pk_{c1}^{-1}.
  The goal of confidentiality is similarly modelled in steps 3.1, 3.2 and 3.3, with the specification that the key pair (pk_{c3}, pk_{c3}^{-1}) is only known by LRA and RA.
  It is worth noticing that valid approaches are present in the literature for the analysis of protocols layered on top of SSL, *e.g.*, [31].
– The formal language specifying the protocol is tailored for describing software systems where all the interactions are made by means of communications. Thus, instead of considering that U physically reaches LRA, a sending operation has been modeled in step 2. Furthermore, to maintain the confidentiality that the physical action of showing the data to an operator should guarantee, a confidential channel has been established by means of pk_{c2}. pk_{c2} is known by all the users (plus the adversary), but only LRA knows pk_{c2}^{-1}.
– In Subsection 5.1 two analyses of the enrollment procedure will be carried out by assuming that an adversary gathers information that should be secret,

i.e., pin_U and SPKAC, respectively. In the respective specification files, the leakage of these information is modeled by sending them on a public channel. Alternatively, it could also be modeled by including them in the initial knowledge of the adversary. Both the formalizations do not alter the result of the analyses. The first choice differs in the sense that one can decide at which point of the computation the adversary will know the secret. The reader is invited to note that this solution allows the analyst to specify in a more natural way different levels of attacks. Indeed, the reception of confidential values over a public channel, at a certain point of the computation, could model an attack over an online server after that those values have been stored in that server.

5 OpenCA analysis

An analysis of the correct issuance of a digital certificate is here performed, with respect to an active adversary that tries to interfere with the enrollment procedure described in Section 4. This analysis is the evolution of preliminary work in the area, presented in [32].

The adversary is able to listen, intercept and forge communication between honest participants. An incorrect certificate is here intended as a certificate that testifies the association between a public key and the owner of a private key that does not correspond to that public key.

In particular, two possible misbehaviours are considered for the analysis. Within the enrollment procedure described in Section 4, it is investigated if it is possible

1. to issue certificates in which the name of an user is tied to the public key whose correspondent private key is only known by another user;
2. to issue certificates in which the public key of an user, that also knows the correspondent private key, is tied to the name of another one (that does not know that private key).

If one of the two alternatives occur, it means that the enrollment procedure leads to the issuance of certificates in which there is not a correct correspondence between the owner of the certificate and the certified public key.

The occurrence of the first alternative could cause a *responsibility* attack, *i.e.*, someone could sign some documents and makes another user responsible for that signature. The failure of the second property could cause a *credit* attack, *i.e.*, someone could claim credit for the origin of a document signed by another user. It is worth noticing that Abadi has formally discussed the properties of responsibility and credit in [33]. Also, [34] further investigates these topics and an

attempt is given to formalize, within the theoretical framework fully presented in [23], some of the examples discussed by Abadi in [33].

First, the initial knowledge of the adversary is set to the set of public messages that it knows at the beginning of the computation, *e.g.*, the names and the public keys of the other participants, its public/private key pair, its identity card.

Thus, the input given to the tool is, informally, as follows (details in Appendix B):

- **Specification file**: OpenCA
- **Formula**:
  $\{name_U, pk_X\}_{pk_{ca}^{-1}}$ *or* $\{name_X, pk_U\}_{pk_{ca}^{-1}}$
- **Initial knowledge**:
  $name_X, pin_X, pk_X, pk_X^{-1}, \{name_X\}_{pk_{Gov}^{-1}}, pk_U, name_U, pk_{c1}, pk_{c2}$.
- The result is **No attack found**.

The specification file is written by following the input language shown in Appendix A. The same appendix contains some excerpts of that specification file.

By requiring the analysis over that particular formula, a computation is searched such that, at the end of such a computation, the adversary knows either message $\{name_U, pk_X\}_{pk_{ca}^{-1}}$ or message $\{name_X, pk_U\}_{pk_{ca}^{-1}}$ (*i.e.*, certificates attesting an incorrect association between a public key and its owner). The result of the analysis gives that such a computation does not exist. Thus, the procedure is correct, with respect to the investigated properties, and at least at the modeled conceptual level.

5.1 Some attacks on the RA server

Under some circumstances, it is possible to force the emission of incorrect certificates. This can happen without breaking CA, but by performing some preliminary attacks on RA. Indeed, the leakage of some confidential information, recorded into RA, may lead to an incorrect certificate issue.

As is common in security analysis, the protocol is investigated by considering that the adversary obtains some information considered confidential among the honest participants.

In particular, here it is supposed that X obtains pin_U and SPKAC by performing a direct attack on the RA server, which is an on-line machine and it is more vulnerable to attacks than the off-line certification authority.

Two attacks can occur, depending on the leaked secret:

- X knows pin_U: then, X can associate its public key to the user's identity.

– X knows SPKAC: then, X can associate its name to the public key of user[1].

First attack. X knows pin_U. The input of PAMOCHSA is:

– **Specification file**: OpenCA_known_pin.
– **Formula**: $\{name_U, pk_X\}_{pk_{ca}^{-1}}$.
– **Initial knowledge**: same as before
– The result is **Attack found**

Indeed, X is able to force the issuing of a certificate where its public key is tied to the name of U. Informally, the attack consists of the following steps (see also Fig. 3). In the picture, step i indicates the normal execution of the protocol by the honest participants, as illustrated in Fig. 1, whereas step i_X indicates that step i is performed by the intruder:

$$
\begin{array}{lll}
1 & U \mapsto ES & : \{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{c1}} \\
1_X & X(U) \mapsto ES & : \{name_U, pk_X, pin_U, \{pk_X, n_X\}_{pk_X^{-1}}\}_{pk_{c1}} \\
2 & U \mapsto LRA & : \{\{name_U\}_{pk_{Gov}^{-1}}, pin_U\}_{pk_{c2}} \\
3.1 & LRA \mapsto RA & : \{name_U, pin_U\}_{pk_{c3}} \\
3.2 & as \quad usual & \\
3.3 & as \quad usual & \\
4 & RA \mapsto CA & : \{name_U, pk_X, pin_U, \{pk_X, n_X\}_{pk_X^{-1}}\}_{pk_{LRA}^{-1}} \\
5 & CA \mapsto RA & : \{name_U, pk_X\}_{pk_{ca}^{-1}} \\
6 & ES \mapsto U & : \{name_U, pk_X\}_{pk_{ca}^{-1}}
\end{array}
$$

Fig. 3. OpenCA - First attack

– Item 1. U connects to ES, as usual.
– Item 1_X. X connects to ES and sends a certificate request consisting of username $name_U$, pin_U it has previously discovered by performing an attack on RA, its public key pk_X and SPKAC, *i.e.*, X's public key plus another nonce n_X, signed by pk_X^{-1}.
– Item 2. U gets in contact with the LRA operator to prove that the data in the request of step 1 are correct, as usual.
– Item 3.1. LRA operator connects to RA Server and approve the request corresponding to the related username and pin_U. However, now two requests exist, both related to the same name and pin. Thus, it could be possible that

[1] This holds because of an incomplete management of SPKAC in the investigated version of OpenCA, see Section 5.2 for details.

the LRA operator approves the wrong request. (Note that steps 3.2 and 3.3 are not formalized in Fig. 3, for the sake of simplicity).
– Steps 4, 5 and 6 are as usual. The final issued certificate ties the identity of the user with X's public key.

The avoidance of this kind of attack should be actuated by enforcing security policies to keep sensitive values (like pin_U) secret.

Second attack X knows SPKAC. Once again, it is supposed that the adversary performs an attack on RA (in the model, it receives this piece of information over a public channel). The input of PAMOCHSA is:

– **Specification file**: OpenCA_known_SPKAC.
– **Formula** $\{name_X, pk_U\}_{pk_{ca}^{-1}}$.
– **Initial knowledge**: same as before.
– The result is **Attack found**.

Informally, the attack consists of the following steps (see also Figure 4):

$$
\begin{array}{lll}
1 & U \mapsto ES & : \{name_U, pk_U, pin_U, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{c1}} \\
1_X & X \mapsto ES & : \{name_X, pk_U, pin_X, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{c1}} \\
2_X & X \mapsto LRA & : \{\{name_X\}_{pk_{Gov}^{-1}}, pin_X\}_{pk_{c2}} \\
3.1 & LRA \mapsto RA & : \{name_X, pin_X\}_{pk_{c3}} \\
3.2 & as \quad usual & \\
3.3 & as \quad usual & \\
4 & RA \mapsto CA & : \{name_X, pk_U, pin_X, \{pk_U, n_U\}_{pk_U^{-1}}\}_{pk_{LRA}^{-1}} \\
5 & CA \mapsto RA & : \{name_X, pk_U\}_{pk_{ca}^{-1}} \\
6_X & ES \mapsto X & : \{name_X, pk_U\}_{pk_{ca}^{-1}}
\end{array}
$$

Fig. 4. OpenCA - Second attack

– Item 1. U connects to ES as usual.
– Item 1_X. X connects to ES and sends its certificate request consisting of $name_X$, a newly created nonce pin_X, the user public key pk_U and the discovered SPKAC of the user.
– Item 2_X. X gets in contact with the LRA operator to prove that the data contained in the request in step 2 are correct. X can physically reach the LRA operator and it can show its identity card.
– Item 3.1. The LRA operator connects to the RA server and it approves the request corresponding to $name_X$ and pin_X.

– Item 4. The certificate requests are exported from the RA server and imported into the CA server, as usual. The CA operator checks if all the requests are signed by the LRA operator and if the Netscape SPKAC is correctly signed. This is done by checking that pk_U is able to verify the signature on SPKAC.
– Item 5. The CA operator issues the certificate and exports it into the RA server.
– Item 6_X. X connects to ES and gets the certificate in which pk_U is tied to X's identity.

5.2 A note on the use of SPKAC

The software architecture investigated above is based on OpenCA v.0.2.0 and OpenSSL v.0.9.4. As explicitly mentioned in the OpenSSL documentation, the SPKAC nonce challenge is not used in that implementation. Thus, the second attack can be avoided by simply recording the nonce previously used in a SPKAC structure.

In order to look for a practical evidence of this drawback, a test Certification Authority has been built, based on the 0.9.4 SSL code. Actually, two requests with the same SPKAC have been generated and, consequently, two certificates have been issued, with different user name but same public key. However, if SPKAC is correctly used, a single CA cannot issue such erroneous certificates.

On the other hand, in a more complex PKI infrastructure this is not true anymore. Let the reader consider a PKI infrastructure consisting of a root CA and two sub-CAs, CA_1, and CA_2 (*i.e.,* a hierarchical structure). The root issues the certificates for its sub-CAs, while these directly interact with the users. Let the reader suppose that user A discovers the SPKAC of user B. B is also supposed to have previously obtained a valid certificate from CA_2. A can send a correct certificate request to CA_1, by using its name and B's SPKAC. As a consequence, A may obtain a valid certificate where its name is tied to B's public key. Under this scenario the nonce is useless, since CA_1 receives the SPKAC for the first time.

A possible solution could be the insertion in the SPKAC structure of the identity of the user. Indeed, the inclusion of the identity is well known to be good practice, [35].

Thus, one can think to follow, *e.g.*, the example of the PKCS#10 structure, that is a standard describing the syntax for making a certificate request. In subsection 6.1 the interested reader will find more about the format and usage of this standard.

Here, it is anticipated that, by following the PKCS#10 standard, the user request contains a digital signature (generated with the user's private key) that

signs the name of the user, plus some additional attributes. Then, it makes no sense to send a duplicate request, as the one in step 1_X, Fig. 4. Such an attack would not work, simply because the LRA operator is now able to check if the name of the user making the request matches the name in the signed field. Updating the protocol by following the new guidelines, step 1_X becomes:

$$X \mapsto ES : \{name_X, pk_U, pin_X, \{name_U, pk_U, attributes\}_{pk_U^{-1}}\}_{pk_{c1}}$$

Thus, X cannot simply reply the request (if not in possess of pk_U^{-1}). Indeed, the operator would find a mismatch between the identity within the signature ($name_U$) and the identity outside the signature ($name_X$).

Upon analyzing the protocol through PAMOCHSA, the test OpenCA version has been updated by the authors of this paper, by allowing only requests*à la* PKCS#10.

6 The SCEP Enrollment Phase

A description of the Simple Certificate Enrollment Procedure (SCEP) is hereafter given. The current section and the following one are based on studies appeared in [1], here revised.

SCEP is a communication protocol whose goal is the secure issuance of certificates to network devices, such as routers and gateways, using existing technology. Up to today, the last document describing SCEP is an Internet Draft available on Internet at [21].

SCEP supports many operations, like CA public key distributions, certificate enrollment and revocation, certification and CRL query. Hereafter, the attention will be mainly focussed on the enrollment phase. That consists of two main phases:

- the user U, identified by a subject name consisting of the Fully Qualified Domain Name (*e.g.*, alice.somewhere.com), asks for a digital certificate. It composes its certificate request and sends it to a Certification Authority Server.
- The Certification Authority tests the correctness of the received request[2]; in case of positive outcome, CA issues the certificate, digitally signs it and sends it to the applicant.

[2] In subsection 6.2 it is explained how CA tests the correctness of the request.

6.1 User Certificate Request

After having obtained the CA's certificate, necessary to retrieve CA's public key in order to enroll, the user generates its request using PKCS#10 and sends it to the CA exploiting PKCS#7. PKCS#10 and PKCS#7 are "de facto" standards, issued by RSA Labs: PKCS#10 describing the syntax for certification requests and PKCS#7 defining formats to represent data with the addition of cryptographic information, *i.e.*, encrypted data or digital signatures. Briefly, a PKCS#10 request can be formalized as follows:

$$PKCS\#10 := \{name_U, pk_U, pin_U, \{name_U, pk_U, pin_U\}_{pk_U^{-1}}\}$$

where $name_U$ is the Subject Name of the user U, pk_U is the public key to be certified and pin_U is a secret that associates the subject name to that certificate request[3].

PKCS#10 is completed by adding a digital signature over the name, public key and pin of the user. This signature acts as a proof of possession, *i.e.*, once CA has verified the signature, it has proof that whoever originated the signature holds the corresponding private key.

Upon composing PKCS#10, U builds the Enveloped Data (encrypted data plus encrypted key by means of RSA)[4], exploiting PKCS#7 technologies:

$$EnvelopedData := \{PKCS\#10\}_{KEY}, \{KEY\}_{pk_{CA}}$$

where KEY is a randomly generated symmetric key. The construction of Enveloped Data provides the encryption of KEY with the public key of the CA, pk_{CA}, so that only CA can retrieve KEY and successfully obtain the PKCS#10 as a clear-text.

To complete the enrollment request, U creates Signed Data (data plus digital signatures), basically consisting of:

$$SignedData := \{EnvelopedData, \{ID, Nonce\}_{pk_U^{-1}}\}$$

ID is the fingerprint of the public key to be certified and its aim is to uniquely identify this transaction. $Nonce$ is a random number generated by U, and its aim is to prove the freshness of the response from the CA to the user

[3] This pin is used for certificate revocation (currently implemented as a manual process: U phones a CA Operator asking for revocation of its certificate, the operator replies asking for the challenge password, and if it coincides with the one contained in the request, the certificate will be revoked). The pin can also be used to authenticate U's identity, as explained in subsection 6.2

[4] For the sake of readability the structures of Enveloped Data, Signed Data and Get Cert Initial message are here simplified (without, however, affecting the results of the analysis).

request. Answers[5] of CA to U's enrollment request can be of three kinds and they all contain the same ID and $Nonce$ as in the user request. In case of a success, CA successfully issues the requested certificate; in case of a pending response, CA is configured to act in manual mode. Before the emission, it has to carry out some checks to verify enrollment request correctness; finally, in case of a failure, CA does not issue the certificate.

If the user receives a pending response, then it can enter into a polling mode, *i.e.*, it can periodically send to CA *Get Cert Initial* messages, pressing for the issuance.

$$GetCertInitial := \{\{name_U, ID\}_{KEY}, \{KEY\}_{pk_{CA}}, \{Nonce\}_{pk_U^{-1}}\}$$

6.2 Modeling the enrollment procedure

In protocols that use public key cryptography, the association between the public keys and the identities with which they are associated must be authenticated in a secure manner. SCEP provides two authentication methods: a manual one and one based on a pre-shared secret.

In manual mode, once a certificate request has been sent to CA, U waits until its identity can be verified using any reliable mechanism, to be performed over channels other than the Internet. This mechanism could be performed, *e.g.*, by personally reaching a devoted operator and directly showing some appropriate credentials, or by delivering such credentials by surface mail or phone. In particular, [21] suggests that CA generates the fingerprint of the PKCS#10 retrieved from the user request and compares it with the one computed by the user itself. During this period, the state of the whole transaction is set to pending.

Otherwise, CA can choose to act in automatic mode: before any request takes place, CA distributes a pre-shared secret to the user (assumed unique for each user). The user will then insert the secret in the request (pin_U). Then, CA should check the correspondence between pin_U and the name included in the PKCS#10.

Enrollment procedure with manual user authentication. The enrollment procedure with manual authentication of the user can be described as follows:

1. U connects to CA and sends enveloped PKCS#10 and authenticated attributes. $Nonce_U$ is inserted in the authenticated attributes to prevent replay attacks (from the user point of view). Thus, the answer from CA to U must contain the same nonce of the previous message from U to CA.

[5] These answers contain the same ID and $Nonce$ present in User Certificate Request.

$1\ U \mapsto CA : \{name_U, pk_U, pin_U, \{....\}_{pk_U^{-1}}\}_{KEY}, \{KEY\}_{pk_{CA}}, \{ID_U, Nonce_U\}_{pk_U^{-1}}$
$2\ CA \mapsto U : \{ID_U, Nonce_U, \text{“}pending''\}_{pk_{CA}^{-1}}$
$3\ U \mapsto CA : \{name_U, ID_U\}_{KEY}, \{KEY\}_{pk_{CA}}, \{Nonce1_U\}_{pk_U^{-1}}$
$4\ CA \mapsto U : Hash\{name_U, pk_U, pin_U, \{....\}_{pk_U^{-1}}\}$
$5\ U \mapsto CA : Comparison : ok/ko$
$6\ CA \mapsto U : \{\{name_U, pk_U\}_{pk_{CA}^{-1}}\}_{KEY1}, \{KEY1\}_{pk_U}, \{Nonce1_U\}_{pk_{CA}^{-1}}$

Fig. 5. SCEP Enrollment Phase with manual user authentication.

2. CA replies with a pending status, same transaction identifier and same nonce as in the user request.
3. U enters into polling mode by periodically sending *Get Cert Initial* messages to CA, until it either receives the certificate or rejection or it simply times out.
4. Communications over channels 4 and 5 should be intended other than the Internet. This reliable communication is intended to be by phone or by surface mail. In any case, CA must communicate to U the fingerprint of the PKCS#10 received in Message 1 (Message 4). Thereafter, the user can compare the fingerprint with the one computed from its original PKCS#10 (Message 5).
5. U gives a positive, or negative, answer to CA, depending on the result of the comparison.
6. Upon receiving a positive answer, CA issues the certificate. SCEP distributes the certificates in an enveloped mode, followed by the same user nonce contained in the previous *Get Cert Initial*.

Enrollment Procedure with Automatic User Authentication. When a pre-shared secret scheme is used, the enrollment procedure is quite simple (Fig. 6). The user authentication is subject to the correspondence between pin_U and $name_U$.

$1\ U \mapsto CA : \{name_U, pk_U, pin_U, \{...\}_{pk_U^{-1}}\}_{KEY}, \{KEY\}_{pk_{CA}}, \{ID_U, Nonce_U\}_{pk_U^{-1}}$
$2\ CA \mapsto U : \{\{name_U, pk_U\}_{pk_{CA}^{-1}}\}_{KEY1}, \{KEY1\}_{pk_U}, \{Nonce_U\}_{pk_{CA}^{-1}}$

Fig. 6. SCEP Enrollment Phase with automatic user authentication.

7 SCEP Analysis

Briefly, the security goals of SCEP are that no adversary can: i) subvert the public key/identity binding from that intended; ii) discover the identity information in the enrollment request and in the issued certificates; iii) cause the revocation of certificates with any non-negligible probability.

The first and second goals are met by exploiting encryption and digital signatures with authenticated public keys. The third goal is met through the use of a challenge password for revocation.

The revocation phase is not a concern to the scope of this paper but rather the phase of enrollment has been considered.

When running the tool, a finite number of processes, each of them having a finite behavior, has been considered. Note that, with regard to this scenario, SCEP guarantees the correct emission of certificates (*i.e.*, goals 1 and 2 are achieved).

However, the attention will be focused on particular checks suggested in [21], in order to understand some security mechanisms and the possible consequences of their absence.

7.1 Relevance of the user authentication phase

Here, a simple analysis is performed, by checking if an intruder could be able to force CA to issue certificates in which the public key/identity binding is subverted.

A specification following the Internet Draft, *i.e.*, the one including the comparison of the PKCS#10 fingerprints, has been checked. The results confirm the correct emission of the certificate.

When no fingerprint comparison takes place, the protocol results vulnerable to a man in the middle attack. The tool automatically unveils the attack. This consists of the following steps (see also Fig. 7):

1. U connects to CA as in a normal execution. The request is intercepted by X.
2. X sends to CA a certificate request containing the user name.
3. CA's answer contains a pending status.
4. U enters into polling mode. Its *Get Cert Initial* message is intercepted by X.
5. X simulates the polling mode.
6. Something went wrong with the comparison of the fingerprint. It is possible to issue the certificate related to the request of X.

The absence of the fingerprint comparison could represent either the fact that CA does not contact the user to communicate the received fingerprint (no Message 4 in Fig. 5) or the fact that the user itself omits the comparison (no Message 5 in Fig. 5).

$1\ U \mapsto X(CA) : \{name_U, pk_U, pin_U, \{...\}_{pk_U^{-1}}\}_{KEY}, \{KEY\}_{pk_{CA}}, \{ID_U, Nonce_U\}_{pk_U^{-1}}$

$2\ X(U) \mapsto CA : \{name_U, pk_X, pin_X, \{name_U, pk_X, pin_X\}_{pk_X^{-1}}\}_{KEY_X}, \{KEY_X\}_{pk_{CA}}, \{ID_X, Nonce_U\}_{pk_X^{-1}}$

$3\ CA \mapsto U : \{ID_X, Nonce_U, \text{“}pending''\}_{pk_{CA}^{-1}}$

$4\ U \mapsto X(CA) : \{name_U, ID_U\}_{KEY}, \{KEY\}_{pk_{CA}}, \{Nonce1_U\}_{pk_U^{-1}}$

$5\ X(U) \mapsto CA : \{name_U, ID_X\}_{KEY_X}, \{KEY_X\}_{pk_{CA}}, \{Nonce1_U\}_{pk_X^{-1}}$

$6\ CA \mapsto U : \{\{name_U, pk_X\}_{pk_{CA}^{-1}}\}_{KEY1}, \{KEY1\}_{pk_X}, \{Nonce1_U\}_{pk_{CA}^{-1}}$

Fig. 7. No fingerprint comparison.

The particular structure of messages in SCEP helps U to discover that it actually receives an incorrect certificate. Indeed, U will not be able to open the envelope received in message 6, since $KEY1$ is encrypted with an unexpected key, pk_X. If the user does not receive any certificate (*e.g.*, the adversary could intercept message 6), it will send to CA a sequence of *GetCertInitial* messages, pressing for the certificate. Possible further interceptions of these messages could then lead to a time out interrupt of U's connection.

7.2 How to avoid the issuance of two identical certificates

The draft encourages CAs to enforce the so called *certificate-name uniqueness*: at any time, there will be only one pair of keys for a given subject name and key usage combination.

The authors of this paper prefer to distinguish between two kinds of uniqueness. Thus, the above mentioned property will be referred as a *weak uniqueness*, meaning that it is not possible to issue two (or more) valid certificates with the same subject name, same public key and key usage whose validity periods overlap. With respect to the validity period of a certificate, weak uniqueness is in contrast with another property, called *strong uniqueness*, meaning that it is *never* possible to issue two (or more) valid certificates with the same subject name, same public key and key usage.

To better distinguish between weak and strong uniqueness, let the reader suppose the existence of two certificates with same subject name, same public key and key usage. Then, it could be the case that their validity periods overlap (*e.g.*, the validity period of the first certificate is January 1st, 2005 – January 1st, 2006, whereas the one of the second certificate is July 1st, 2005 – July 1st, 2006). It could be also the case that their validity periods do not overlap at all (*e.g.*, January 1st, 2005 – January 1st, 2006 and July 1st, 2006 – July 1st, 2007).

In the first case, both weak and strong uniqueness do not hold. In the last case, it holds weak uniqueness, but not the strong one.

If CA issues two identical certificates, they will however differ in the serial number, say $sn1$ and $sn2$. Let the reader consider that their validity periods overlap. Suppose also that an adversary was able to force CA to issue the last certificate, so that the adversary is conscious of its existence, while U is not. There are multiple reasons for preventing the re-transmission of the same data from creating a second certificate. The most significant reasons are:

- considering a large scale application scenario, the computational cost in generating and signing unused certificates is high;
- a document digitally signed with the private key corresponding to U's public key could be valid longer than expected, because the unexpired certificate would validate the signature;
- U could maliciously extend her/his own certificate's validity even when (s)he is purposely denied the right to a new certificate; *e.g.*, in a corporate environment, an employee might have access to a certain facility but only for a limited time.

Each public key to be certified is strictly connected to the Transaction Identifier ID, since it is the fingerprint of the key. To guarantee weak uniqueness, it is assumed that CA records the pair $(name_U, ID_U)$. A specification expecting weak uniqueness (*i.e.*, including that record) has been checked. The analysis confirms that it is not possible to issue two identical certificates whose validity periods overlap.

However, if CA neglects to record that pair, the protocol results vulnerable to the following attack: the intruder eavesdrops on a legitimate certificate request and it simply repeats it later. This replay attack is reported in Fig. 8.

1. U connects to CA, as usual. Its request is eavesdropped by X.
2. X repeats the request, pretending to be U.
3. CA issues a first valid certificate with serial number sn_1.
4. CA issues a second valid certificate with serial number sn_2, because it omits checks on crucial fields of the requests. X intercepts this message.

The user itself may unconsciously contribute towards the issuance of identical certificates. Indeed, if U times out, CA could issue a first certificate but U may not receive anything. Consequently, U is allowed to re-issue the same request and the absence of checks by CA may lead to the issuance of a double certificate.

Finally, when automatic enrollment is used, weak uniqueness is not enough to protect against replay attacks on expired certificates requests. Indeed, X could

$1\ U \mapsto CA \quad : \{name_U, pk_U, pin_U, \{...\}_{pk_U^{-1}}\}_{KEY}, \{KEY\}_{pk_{CA}}, \{ID_U, Nonce_U\}_{pk_U^{-1}}$
$2\ X(U) \mapsto CA : \{name_U, pk_U, pin_U, \{...\}_{pk_U^{-1}}\}_{KEY}, \{KEY\}_{pk_{CA}}, \{ID_U, Nonce_U\}_{pk_U^{-1}}$
$3\ CA \mapsto U \quad : \{\{name_U, pk_U, sn_1\}_{pk_{CA}^{-1}}\}_{KEY1}, \{KEY1\}_{pk_U}, \{Nonce_U\}_{pk_{CA}^{-1}}$
$4\ CA \mapsto X(U) : \{\{name_U, pk_U, sn_2\}_{pk_{CA}^{-1}}\}_{KEY2}, \{KEY2\}_{pk_U}, \{Nonce_U\}_{pk_{CA}^{-1}}$

Fig. 8. Replay attack

eavesdrop on a legitimate certificate request. The automatic procedure will lead to the issuance of $\{name_U, pk_U\}_{pk_{ca}^{-1}}$. Then, X may send the replay of the request once that first certificate has expired. This can cause a new certificate to be issued with the same subject name - public key binding. The existence of the last certificate may cause a document previously signed by U to be valid longer than expected. It is opinion of the authors of this paper that, to avoid such a vulnerability, CAs should guarantee the strong uniqueness property.

8 Conclusions

In this paper (part of) the OpenCA and the SCEP enrollment phases have been formalized. An analysis of some of their properties has been performed by means of a software tool. With regard to a limited scenario (finite number of processes with a finite behavior), no attack has been found.

With regard to OpenCA, it has been shown that, when sensitive data are not accurately stored, the leakage of those data may lead to an incorrect certificate issuance. This can cause attacks on responsibility and credit, *e.g.*, by making a signature to be considered valid when it should not, or by considering someone responsible for something (s)he has not signed.

As far as SCEP is concerned, when automatic enrollment is used, a vulnerability was found, concerning a replay attack on expired certificates requests. Furthermore, some checks suggested in the SCEP draft have been purposely omitted. As a consequence, a subversion of the correct public key/identity binding and the emission of duplicate valid certificates are detected.

This paper has focused on aspects of certificate delivery. Nevertheless, problematics related to digital certificates are wide, and future work should be devoted to the analysis of problems such as certificate revocation.

Acknowledgements

Work partially supported by MIUR project: “Strumenti, Ambienti e Applicazioni Innovative per la Società dell’Informazione”, subproject SP1: Reti INTERNET:

“efficienza, integrazione e sicurezza”; by MIUR project: “Tecniche e Strumenti Software per l’Analisi della Sicurezza delle Comunicazioni in Applicazioni Telematiche di Interesse Economico e Sociale”; by CNR project: “Strumenti, Ambienti ed Applicazioni Innovative per la Società dell’Informazione”; by “MEFISTO”: Metodi Formali per la Sicurezza ed il Tempo; by CSP projects: “SeTAPS I” and “SeTAPS II”; by Quality of Protection (QoP) project: “CREATE-NET”.

The authors would like to thank the anonymous referees for their significant advice.

References

[1] Martinelli F, Petrocchi M, Vaccarelli A. Automated analysis of some security mechanisms of SCEP. *Proc. of ISC’02*, volume 2433 LNCS, pages 414–427. Springer, 2002.

[2] Abadi M, Gordon AD. Reasoning about cryptographic protocols in the Spi Calculus. *Proc. of CONCUR’97*, volume 1243 LNCS, pages 59–73. Springer, 1997.

[3] Focardi R, Gorrieri R, Martinelli F. Non interference for the analysis of cryptographic protocols. *Proc. of ICALP’00*, volume 1853 LNCS, pages 354–372. Springer, 2000.

[4] Marchignoli D., Martinelli F. Automatic verification of cryptographic protocols through compositional analysis techniques. *Proc. of TACAS’99*, volume 1579 LNCS, pages 148–162. Springer, 1999.

[5] Meadows C. Formal verification of cryptographic protocols: a survey. *Proc. of ASI-ACRYPT’94: Advances in Cryptology*, volume 917 LNCS, pages 135–150. Springer, 1995.

[6] Lowe G., Roscoe AW. Using CSP to detect errors in the TMN protocol. *Software Engineering*, **23**(10):659–669, 1997.

[7] Shmatikov V, Stern U. Efficient finite state analysis for large security protocols. *Proc. of CSFW’98*, pages 105–116. IEEE Computer Society Press, 1998.

[8] Thayer FJ, Herzog JC, Guttman JD. Strand Spaces: proving security protocols correct. *Journal of Computer Security*, **7**(1):191–230. 1999.

[9] Lowe G. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Proc. of TACAS’96*, volume 1055 LNCS, pages 147–166. Springer, 1996.

[10] Mitchell JC, Mitchell M, Stern U. Automated analysis of cryptographic protocols using Murphi. *Proc. S&P’97*, pages 141–153. IEEE Computer Society Press, 1997.

[11] Roscoe AW, Goldsmith MH. The perfect spy for model-checking crypto-protocols. booktitle = *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[12] Abadi M, Tuttle MR. A semantics for a logic of authentication. *Proc. SPDC’91*. pages 201–216. ACM, 1991.

[13] Kindred D, Wing JM. Fast automatic checking of security protocols. *Proc. 2nd Usenix Workshop on Electronic Commerce*. pages 41–52. 1996.

[14] Paulson LC. Proving properties of security protocols by induction. *Proc. CSFW’97*. pages 70–83. IEEE computer Society Press, 1997.

[15] Abadi M. Secrecy by typing in security protocols. *Journal of the ACM*. **46**(5):749–786. ACM, 1999.

[16] Bodei C, Degano P, Nielson F, Nielson HR. Static analysis for the pi-Calculus with applications to security. *Information and Computation*. **168**(1):68–92. 2001.

[17] Ryan P, Schneider S, Goldsmith M, Lowe G, Roscoe B. *The modelling and analysis of security protocols: the CSP approach*. Addison-Wesley, 2002.

[18] Needham R, Schroeder M. Using encryption for authentification in large networks of computers. *Communications of the ACM*. **21**(12). ACM, 1978.
[19] Lowe G. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*. **56**(3):131–136. 1995.
[20] Housley R, Ford W, Polk W, Solo D. RFC 2459: Internet X.509 public key infrastructure certificate and CRL profile. The Internet Society. IETF, 1999. http://www.ietf.org/rfc/rfc2459.txt. Last access March 30, 2005.
[21] Liu X, Madson C, McGrew D, Nourse A. Internet Draft: draft-nourse-scep-11, Cisco Systems, 2005. http://www.vpnc.org/draft-nourse-scep. Last access March 30, 2005.
[22] Cichocki T, Gorski J. Formal support for fault modeling and analysis. *Proc. of SAFECOMP'01*, volume 2187 LNCS, pages 190–199. Springer, 2001.
[23] Martinelli F. Analysis of security protocols as open systems. *Theoretical Computer Science* **290**(1): 1057-1106. 2003.
[24] Milner R. *Communication and Concurrency*. Prentice Hall, 1989.
[25] Andersen HR. Partial model checking. *Proc. of 10th Symposium Logic in Computer Science*. pages 398–407. IEEE Computer Society Press. 1995.
[26] Dolev D., Yao A. On the security of public key protocols. *Trans. Inf. Theory*. **29**(2):198–208. IEEE Computer Society Press. 1983.
[27] The PAMOCHSA homepage. http://www.iit.cnr.it/staff/fabio.martinelli/pamochsa.htm. Last access March 30, 2005.
[28] Lowe G. Towards a completeness result for model checking of security protocols. *Proc. of CSFW'98*, pages 96–105. IEEE Computer Society Press. 1998.
[29] Stoller S. A reduction for automated verification of authentication protocols. *Technical Report 520*. Computer Science Dept., Indiana University, 1998.
[30] Freier AO, Karlton P, Kocher PC. The SSL Protocol Version 3.0 - Internet Draft http://wp.netscape.com/eng/ssl3/ssl-toc.html, 1996. Last access March 30, 2005.
[31] P. Broadfoot and G. Lowe. On distributed security transactions that use secure transport protocols. In *Proc. of CSFW'03*, pages 141–155. IEEE, 2003.
[32] Giani A, Martinelli F, Petrocchi M, Vaccarelli A. A case study with PamoChSA: a tool for the automated analysis of protocols.. *Proc. of SCI-ISAS'01*, volume 5, pages 203–210. IIIS, 2001.
[33] Abadi M. Two facets of authentication. *Proc. of CSFW'98*, pages 27–32. IEEE Computer Society Press, 1998.
[34] Gorrieri R, Martinelli F, Petrocchi M. A formalization of credit and responsibility. *Proc. of SASYFT'04*. *Technical Report LIFO 2004-11*. 2004.
[35] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering* 22(1): 6–15. 1996.

A The PAMOCHSA input language

This appendix describes the syntax of the PAMOCHSA input language and gives examples for the drawing up of an input file.

A.1 Typed messages

The messages are typed, *i.e.*, each message has an associated type that denotes its structure. Types are used to record the structure of the exchanged data. Since

certain operations are meaningful only over data with a certain structure, types permit to define managing rules that precisely correspond to those operations.

A message *m* of type *T* is represented as: *m : T*. This expression forms a typed message. Typed messages can be basic or compound and they are recursively defined.

A.2 Grammar of the input language

The input file of PAMOCHSAis a so called *experiment* file with the following structure.

< FORMULA >
Formula
< /FORMULA >

< KNOWLEDGE >
Initial Knowledge
< /KNOWLEDGE >

<HIDE_CHANNELS >
Hidden Channels List
< /HIDE_CHANNELS >

< SPEC >
Protocol Specification
< /SPEC >

Special identifiers contain the four sections constituent the file.

Each formula can be either a single typed message or a set of typed messages tied by the logic operators *and*, *or* and *not*.

The initial knowledge is a set of typed messages.

The list of hidden channels is a list of channels over which the intruder cannot interfere during the run of the protocol.

The protocol specification is actually the body of the protocol, *i.e.*, a sequence of sending, reception and control actions.

In particular, the grammar of the input language is recursively defined as follows (only a simplified version is here reported):

$$
\begin{aligned}
experiment\_form ::= & < \textit{FORMULA} > \quad form \quad < \textit{/FORMULA} > \\
& < \textit{KNOWLEDGE} > \quad m\_list \quad < \textit{/KNOWLEDGE} > \\
& < \textit{HIDE\_CHANNELS} > \quad str\_list \quad < \textit{/HIDE\_CHANNELS} > \\
& < \textit{SPEC} > \quad term \quad < \textit{/SPEC} >
\end{aligned}
$$

$term$ ::= *0*
| *Send* *(* $pstr$, $expr$*).*$term$
| *Recv* *(* $pstr$, $ident$: msg_type*).* $term$
| *If* *(* $expr$ = $expr$*)* *Then* $term$ *Else* $term$ *End* *If*
| *If* *Deduce* *(* $ident$ = $expr$*)* *Then* $term$ *Else* $term$ *End* *Deduce*
| *Choice* c_list *End* *Choice*
| *Parallel* p_list *End* *Parallel*

$expr$ ::= $typed\_msg$
| $ident$
| *Fst* $expr$
| *Snd* $expr$
| *(* $expr$*)*
| *(* $expr$, $expr$*)*
| *Encrypt* *(* $expr$, $expr$*)*
| *Decrypt* *(* $expr$, $expr$*)*

$form$ ::= $typed\_msg$
| $form$ & $form$
| $form$ | $form$
| *Not* $form$
| *(* $form$ *)*

$pstr$ and $ident$ are alphanumeric strings (plus special characters'_' '<' '>' '/').

$form$ can be either a single typed message or a combination of typed messages by means of the logical operators & (AND), | (OR) and Not.

The definition of $term$ maps a set of atomic actions into the tool input language. Indeed,

- 0 is the process that does nothing.

- *Send* *(* $pstr$,$expr$*).*$term$ is the process that can perform the sending of message $expr$ on channel $pstr$ and then it behaves like $term$.

- *Recv* *(* $pstr$,$ident$:msg_type*).*$term$ is the process that receives a message $ident$ of type msg_type on channel $pstr$ and then it behaves like $term$;

- *If* *(* $expr$ = $expr$*)* *Then* $term$ *Else* $term$ *End* *If* maps the match construct of CryptoCCS.

- *If* *Deduce* *(* $ident$ = $expr$*)* *Then* $term$ *Else* $term$ *End* *Deduce* is the inference construct. The inference system adopted in the paper is shown in Fig. A.2, where rules to perform encryption, decryption and for retrieving the elements of a pair are given. In particular, rule (1) builds the pair of two messages; rules (2) and (3) are used to obtain the elements of a pair; rules (4) and (5) allow messages to be encrypted using a public key of type *EKey* or a private key of type *DKey*; rules (6) and (7) allow messages to be decrypted using the corresponding inverse keys. It is worth noticing that other inference systems are allowed and that the analysis sketched in Section 2 is parametric with respect to the given system.

- *Choice* c_list *End* *Choice* maps the non deterministic choice. Thus, it represents a process that non-deterministically decides to behave as one of the terms in c_list.

- *Parallel* p_list *End Parallel* represents the parallel composition of the processes in p_list.

p_list is a list of terms to be executed in parallel. c_list is a list of terms of which only one will be executed.

A typed message has the following structure:

$$typed\_msg ::= msg : msg\_type;$$

where

$$\begin{array}{lll} msg ::= pstr & & msg\_type ::= ident \\ \quad | \quad (msg) & & \quad | \quad EKey \\ \quad | \quad msg, \quad msg & & \quad | \quad DKey \\ \quad | \quad Enc \quad [pstr]\ (msg) & & \quad | \quad (msg\_type) \\ & & \quad | \quad msg\_type \quad * \quad msg\_type \\ & & \quad | \quad Enc(msg\_type \quad * \quad msg\_type) \end{array}$$

Commas are used to separate elements in a pair, while * is for separating the types of the pair.

Types are freely assigned, except two special types denoting encryption and decryption keys, *i.e.*, $EKey$ and $DKey$, respectively. Public keys are always of type $EKey$, whereas $DKey$ is the type for private keys. The correspondence between a public and a private key is established by the name of those keys, *i.e.*, : *key_A* : *EKey* denotes the public key of A; *key_A* : *DKey* denotes the private key of A.

Symmetric cryptography can be simulated by using a pair of public/private key.

In the recursive definition of $Expr$, Fst (resp., Snd) returns the first (resp., the second) element of a pair, while $Encrypt$ (resp., $Decrypt$) returns the encryption (resp., the decryption) of the first element with the second one, that must be an encryption (resp., a decryption) key.

Message *Enc [pstr] (msg)* can be used to set up formulas. Let the reader suppose that the formula is a typed message consisting of a pair, *e.g.*, (*name*, *nonce*) encrypted with the public key *pkey*. Syntactically, this corresponds to: Enc[pkey](name, nonce) : Enc((Name * Nonce) * EKey).

B OpenCA specification

Here, an excerpt of an experiment file given as input to the tool are presented. In particular, the excerpt is from the experiment file OpenCA describing the enrollment procedure presented in Section 4, Fig. 1.

$$\frac{x : T_1 \; y : T_2}{x, y : T_1 * T_2}(1) \qquad \frac{(x, y) : T_1 * T_2}{x : T_1}(2) \qquad \frac{(x, y) : T_1 * T_2}{y : T_2}(3)$$

$$\frac{x : T \; y : EKey}{Encrypt(x, y) : Enc(T * EKey)}(4) \qquad \frac{x : T \; y : DKey}{Encrypt(x, y) : Enc(T * DKey)}(5)$$

$$\frac{Encrypt(x, y) : Enc(T * EKey) \; y : DKey}{x : T}(6) \quad \frac{Encrypt(x, y) : Enc(T * DKey) \; y : EKey}{x : T}(7)$$

Fig. 9. An example inference system

B.1 OpenCA experiment file

```
<FORMULA>
Enc[pk_ca](u_name,pk_x) : Enc((Name * EKey) * DKey) |
Enc[pk_ca](x_name,pk_u) : Enc((Name * EKey) * DKey)
</FORMULA>

<KNOWLEDGE>
pk_c1 : EKey; pk_c2 : EKey; x_name : Name; x_pin : Pin;
pk_x : EKey; pk_x : DKey; Enc[pk_gov](x_name):
Enc(Name * DKey); pk_u : EKey; u_name : Name
</KNOWLEDGE>

<HIDE_CHANNELS>
c4,c5
</HIDE_CHANNELS>

<SPEC>
Parallel

(* User *)
 Send(c1, Encrypt((((u_name : Name, pk_u : EKey ), u_pin : Pin),
          Enc[pk_u](pk_u, n_u):Enc((EKey * Nonce)* DKey)),pk_c1 : EKey)).
 Send(c2, (Enc[pk_gov](u_name), u_pin) : (Enc(Name * DKey)) * Pin).
 Recv(c6, Y : (Enc( (Name * EKey) *  DKey))).0

And

(* CA  *)
 Recv(c4, Z : Enc ((((Name * EKey) * Pin) *
              Enc ((EKey * Nonce) * DKey)) * DKey)).
 If Deduce ( Z1 = Decrypt (Z, pk_lra : EKey)) Then
                          (* verify LRA signature *)
     If Deduce ( M = Snd (Fst (Z1))) Then
                          (* retrieve user public key *)
        If Deduce (M1 = Decrypt (Snd (Z1), M )) Then
               (* decrypt SPKAC with user public key *)
           If (M = Fst(M1)) Then
                        (* equalities of public keys *)
```

```
              If Deduce (Z2 = Fst (Z1)) Then
                              (* retrieve name_u, pk_u *)
                 Send(c5, Encrypt (Z2 , pk_ca : DKey)).0
                              (* release certificate *)
              End Deduce
           End If
        End Deduce
     End Deduce
 End Deduce

And
(* Enrollment Server *)
[...]

And
(* RA  Server *)
[...]

And
(* LRA Operator *)
  Recv(c2, Z : Enc ((Enc(Name * DKey) * Pin) * EKey)).
   If Deduce (X = Decrypt (Z , pk_c2 : DKey)) Then
    If Deduce (Z1 = Decrypt (Fst X , pk_gov : EKey)) Then
                      (* verify the identity card *)
     Send(c31, Encrypt ((Z1, Snd X), pk_c3 : EKey)).
                              (* name and pin to RA *)
     Recv(c32, Z2 : Enc ((((Name * EKey) * Pin) *
                    Enc((EKey * Nonce) * DKey)) * EKey)).
        If Deduce (Z2_p = Decrypt (Z2 , pk_c3 : DKey)) Then
           If (Z1 = (Fst (Fst (Fst Z2_p)))) Then
                                  (* check name_u *)
            If (Snd X = (Snd (Fst Z2_p))) Then
                                  (* check pin_u *)
             Send(c33, Encrypt (Encrypt (Z2_p, pk_lra : DKey),
                       pk_c3: EKey)).0
            End If
           End If
        End Deduce
    End Deduce
   End Deduce

End Parallel
</SPEC>
```